CS261 Homework 4

Nicholas Dufour, Marielos Sanson, Melvin Low
ndufour, msanson, mwlow

May 30, 2015

1 Problem 19

(a) **Proof.** By linearity of expectation, we know that the expectation of the sum is the sum of the expectations, therefore:
$\text{E}[\sum_{j=1}^{m} c_j x_j] = \sum_{j=1}^{m} \text{E}[c_j x_j]$
In each iteration of the second step of the algorithm, the expected cost of the output increases by at most **x***:
$\sum_{j=1}^{m} \text{E}[c_j x_j] = \sum_{j=1}^{m} c_j \text{E}[x_j] = \sum_{j=1}^{m} c_j \text{P}[S_j \in C_t] = \sum_{j=1}^{m} c_j x_j^* = \mathbf{x^*}$
Since **x*** is a fractional solution, it can only be better than OPT, therefore $\mathbf{x^*} \leq OPT$. There are 2 ln n iterations, therefore the expeCted cost outputed by the algorithm will be:
2 ln n $\mathbf{x^*} \leq$ 2 ln n OPT.

□

(b) **Proof.** Let the element $i \in U$ be contained in k subsets. The probability that i will be covered is equal to:
$P[i$ belongs to a set of $C_t]$ which is equal to 1 - $P[i$ does not belong to a set of $C_t]$. We know that set S_j is selected with probability x_j^* and we know that the sum of the probability of all k subsets that contain i is ≥ 1
$\sum_{j:i \in S_j} x_j^* \geq 1$
Given this, the probability that i is covered is minimized when all the x_j^*'s are equal, such that
$x_j^* = \frac{1}{k}\ \forall j : i \in S_j$
$P[i$ does not belong to a set of $C_t]$ is equal to the probability that none of the k subsets that contain i are chosen. We express this as:
$\prod_{j:i \in S_j} (1 - x_j^*)$
Since we know that for all y, $(1 - y) \leq e^{-y}$, then we can simplify the expression above to:
$\prod_{j:i \in S_j} (1 - x_j^*) \leq (e^{-x_j^*})^k = e^{-1} = \frac{1}{e}$
Since we want $P[i$ belongs to a set of $C_t]$, we know that this is:
$1 - P[i$ does not belong to a set of $C_t] \geq 1 - \frac{1}{e}$ □

(c) **Proof.** P[set cover] $= 1 -$ P[at least one element $i \in U$ not covered]
Since there are 2 ln n iterations in step 2 of our algorithm,
P[$i \in U$ not covered] $=$ P[i belongs to a set $C_t]^{2 \ln n}$
P[$i \in U$ not covered] $\leq \frac{1}{e}^{2 \ln n} = \frac{1}{n^2}$
Since there are a total of n elements in U,
P[at least one element $i \in U$ not covered] $\leq \frac{n}{n^2} = \frac{1}{n}$
P[set cover] $\geq 1 - \frac{1}{n}$ □

(d) We convert our ”Monte Carlo” algorithm into a ”Las Vegas” algorithm. We do so by running the ”Monte Carlo” algorithm and checking if the result produced by it is a set cover. If it is, our algorithm (”Las Vegas”) returns the set cover. Otherwise, it runs another round of ”Monte Carlo” and checks the output, until it finds a set cover. Assuming that the running time of the ”Monte Carlo” algorithm is $T(n)$, the running time of our ”Las Vegas” algorithm will be $R(n) = i[T(n) + f(n)]$, where $f(n)$ is the time it takes to check if the output is a set cover ($O(mn)$ where m = total number of elements and n = total number of sets) and i is the number of rounds we must run before the algorithm returns a feasible set cover. The expected running time is therefore,

$$\begin{aligned}\mathrm{E}[R(n)] &\leq \mathrm{E}[i(T(n) + f(n))] \\ &= (T(n) + f(n))\mathrm{E}[i]\end{aligned}$$

We now calculate $\mathrm{E}[i]$

$$\mathrm{E}[i] = \sum_{k=1}^{\infty} k(1-p)^{k-1}p = \frac{1}{p}$$

Where

$$p = \mathrm{P}[\text{output is set cover}] \geq 1 - \frac{1}{n}$$

Therefore,

$$\mathrm{E}[R(n)] \leq \frac{T(n) + f(n)}{p}$$

$$\mathrm{E}[R(n)] \leq \frac{T(n) + f(n)}{1 - \frac{1}{n}}$$

which means our algorithm runs in polynomial time in expectation.

2 Problem 20

(a) The minimum multiway cut problem can be seen as a special case of the uniform labeling problem as follows:

1. Set the cost of assigning a label to be 0.
2. Label a vertex by assigning it to one of the terminals.
3. Minimize the cost of the edges whose endpoints are assigned to different terminals.

Removing all edges with endpoints assigned to different terminals will result in a feasible solution to the multiway cut problem. Thus, minimizing the cost will result in the optimal solution to the minimum multiway cut problem.

(b) An integer formulation for the problem can be described by changing the last two constraints to:

1. $z_e^i \in \{0, 1\}$
2. $x_v^i \in \{0, 1\}$

where x_v^i is 1 if vertex v is given label i, and 0 otherwise; z_e^i is 1 if edge e contains exactly one endpoint assigned label i, and 0 otherwise. With these definitions, it can be seen that the first operand to the sum is the total cost of edges with distinct labels, and the second operand is the total cost of assignment. This precisely describes the uniform labeling problem. The constaints above are a stricter version of the contraints in the linear programmming relaxation; clearly, a solution to the above will also result in a solution to the linear programming relaxation.

Notes: z_e^i is 1 if the edge e has exactly one vertex labeled with i, otherwise 0. x_v^i is 1 if vertex v is assigned label i, otherwise 0. First sum denotes the sum of the cost of edges with endpoints assigned different labels. Second sum denotes the sum of the cost of assignments. Main constraint is that each vertex can only have one label. Second and third constraints says that z is equivalent to the absolute value of $x_u - x_v$ Fourth constraint says that the edge cost is nonnegative? Fifth constraint says that assignment cost in nonnegative.

(c)

(d)

(e)

(f)

(g)

3 Problem 21

(a) Consider an algorithm that randomly partitions the vertices into k distinct groups, and severs all edges e that go between these two groups. We can show that this achieves at least $\frac{k-1}{k}$ times the optimal. We begin by calculating some expectations. The expected number of edges between any two nodes (a number between 0 and 1) is given by:

$$\mathbb{E}[edge] = \frac{|E|}{|V|(|V|-1)}$$

Every vertex will have to sever edges (if any exist) for all vertices not in the group, of which there are expected to be

$$|V| - \frac{|V|}{k}$$

Consequently, each vertex contributes an expected

$$\left(|V| - \frac{|V|}{k}\right)\frac{|E|}{|V|(|V|-1)}$$

edges to the cut. Simplifying:

$$\left(1 - \frac{1}{k}\right)\frac{|E|}{(|V|-1)}$$

$$\frac{|E|(k-1)}{k(|V|-1)}$$

Let W be the sum of all edge weights:

$$W = \sum_{e \in E} w_e$$

The expected weight contribution of a single vertex is then

$$\frac{W}{|E|}\frac{|E|(k-1)}{k(|V|-1)}$$

$$W\frac{(k-1)}{k(|V|-1)}$$

Since there are $|V|$ vertices total, the expected weight of the cut is:

$$W\frac{|V|(k-1)}{k(|V|-1)}$$

In order for the $\frac{k-1}{k}$ relation to hold, the following relation on the optimal weight O must hold:

$$O \leq \frac{k}{k-1} W \frac{|V|(k-1)}{k(|V|-1)}$$

$$O \leq W \frac{|V|}{|V|-1}$$

Since

$$1 < \frac{|V|}{|V|-1}$$

for all $|V| \geq 1$, we have that

$$W \leq W \frac{|V|}{|V|-1}$$

and since clearly $O \leq W$, our algorithm holds with the appropriate bound.

(b) We begin by observing that a vertex will be included in S if none of its neighbors precede it in the sequence. If a vertex is at position i, and has neighbors N then the probability of none of the neighbors preceding it is equal to the ratio of the number of ways to select $i-1$ elements from $V \setminus N$ to the number of ways to select $i-1$ elements from $V \setminus N$. i.e.,

$$P\{i \in S\} = \frac{\binom{|V \setminus N|}{i-1}}{\binom{|V|}{i-1}}$$

If we let $m = |V|$, assume $\Delta = N$, and (for simplicity) let $j = i-1$, then we have:

$$P\{i \in S\} = \frac{\frac{(m-\Delta)!}{(m-j-\Delta)!j!}}{\frac{(m)!}{(m-j)!j!}}$$

or

$$P\{i \in S\} = \frac{(m-d)!(m-j)!}{m!(m-\Delta-j)!}$$

And the expected number of vertices in S is given by:

$$\mathbb{E}[|S|] = \sum_{i \leq m-\Delta} P\{i \in S\}$$

$$\mathbb{E}[|S|] = \sum_{j \leq m-\Delta+1} \frac{(m-\Delta)!(m-j)!}{m!(m-\Delta-j)!}$$

$$\mathbb{E}[|S|] = \frac{m+1}{\Delta+1}$$

$$\mathbb{E}[|S|] = \frac{|V|+1}{\Delta+1}$$

Let $W = \sum_{v \in V} w_v$. The expected value of a random ordering is therefore:

$$\mathbb{E}[weight] = \frac{|V|+1}{\Delta+1}\frac{W}{|V|}$$

The actual optimal value cannot be more than $\Delta+1$ times this expectation, and indeed multiplying our expectation by this factor yields

$$(|V|+1)\frac{W}{|V|}$$

which is greater than the maximum possible value of W.

(c) This problem is reducible to the probability that an element in a permutation lies between two other given elements. Our algorithm operates by returning a random permutation of all the elements. For any three elements, there are six possible configurations $(i, j, k), (i, k, j), ...$ Observe that for a particular constraint, exactly two of these orderings are valid (since all that matters is that the middle element remains in the middle). Thus the probability of any one constraint being satisfied is $1/3$, and by the linearity of expectation the expected number of satisified constraints is also $1/3$.

4 Problem 22

(a) No response required.

(b) If a dominating set DOM of size k can be computed for G_D, we may set $S = DOM$, and observe that the maximum distance from any vertex to its nearest neighbor is no more than $2D$. Adding in edges with *greater* weights to obtain G will not change this value, and so the solution to the dominating set is also a solution to the metric k-means problem with objective function no more than $2D$. Since we assume DOM can be efficiently computed, so can S be efficiently computed.

(c) The algorithm successfully computes a size-k dominating set because it never selects more than one vertex from the same cluster. To see this, consider that every vertex in any given cluster must be connected to that cluster's center by a an edge of weight no more than OPT. By the triangle inequality, every vertex in the cluster must therefore by connected to every other vertex in the cluster by an edge of weight no more than $2OPT$, and hence these edges are also included in G_{OPT}. Therefore, every vertex in a given OPT cluster must be connected to every other vertex, and so as soon as any vertex from a cluster is selected, every other vertex in that cluster is its neighbor.

(d) We may proceed by:

```
(a) For every edge $e \in G$, sorted by $e_w$ in ascending order (at most $\frac{1}{2}n(n-1)$ operations):
      i. Remove all edges with weight greater than $e_w$ (at most $n(n-1)$ operations).
     ii. Attempt to find a dominating set via the greedy algorithm. If $k$ edges have been added to the dominating
         set already, terminate. (at most $n^2(n-1)$ operations).
    iii. If a dominating set is found, return $2e_w$.
```

There are $\frac{1}{2}n(n-1)$ possibilities for OPT. For each OPT, we have to delete at most n^2 edges. To check that a vertex has no more neighbors in in the dominating set requires checking at most n neighbors to see if they are equal to any of at most n vertices in the dominating set.

(e)

5 Problem 23

(a)

(b)

(c)

(d)

6 Problem 24

(a) If matched correctly, in every instance of the problem we have $OPT = n$ because every i^{th} right-side node is matched to the last $n - i + 1$ left-side nodes. This means that every right-side node can be matched to a different left-side node.

(b)

(c)

(d)

(e)